Chapter 3
Conceptual Modeling

Conrad Bock, Fatma Dandashi, Sanford Friedenthal, Nathalie Harrison, Steven Jenkins, Leon McGinnis, Janos Sztipanovits, Adelinde Uhrmacher, Eric Weisel and Lin Zhang

3.1 Introduction

Over the past decade in the modeling and simulation community, there has been a growing interest in and concern about "conceptual modeling." Generally accepted as crucial for any modeling and simulation project addressing a large and complex

C. Bock (✉)
National Institute of Standards and Technology, Gaithersburg, USA
e-mail: conrad.bock@nist.gov

F. Dandashi
The MITRE Corporation, Mclean, USA
e-mail: dandashi@mitre.org

S. Friedenthal
SAF Consulting, Reston, USA
e-mail: safriedenthal@gmail.com

N. Harrison
Lawrence Livermore National Laboratory, Livermore, USA
e-mail: Nathalie.Harrison@drdc-rddc.gc.ca

S. Jenkins
NASA Jet Propulsion Laboratory, Pasadena, USA
e-mail: sjenkins@jpl.nasa.gov

L. McGinnis
Georgia Institute of Technology, Atlanta, USA
e-mail: leon.mcginnis@isye.gatech.edu

J. Sztipanovits
Vanderbilt University, Nashville, USA
e-mail: janos.sztipanovits@vanderbilt.edu

A. Uhrmacher
University of Rostock, Rostock, USA
e-mail: lin@informatik.uni-rostock.de

R. Fujimoto et al. (eds.), *Research Challenges in Modeling and Simulation for Engineering Complex Systems*, Simulation Foundations, Methods and Applications, DOI 10.1007/978-3-319-58544-4_3

problem, conceptual modeling is not well defined, nor is there a consensus on best practices. “Important” and “not well understood” would seem to qualify conceptual modeling as a target for focused research.

One may define conceptual models as “early stage” artifacts that integrate and provide requirements for a variety of more specialized models. In this view, conceptual models provide a foundation from which more formal and more detailed abstractions can be developed and eventually elaborated into analysis models (e.g., for simulation). However, “early” and “late” are relative terms that apply within each stage of development. For example, creating an analysis model might involve describing (i.e., modeling) the analysis independently of software (“conceptually”) before implementation and execution. As a consequence, there might be multiple “early” models: conceptual models of reality and conceptual models of analysis; and there may be multiple versions of conceptual models as the understanding of the target system matures and the analysis design and implementation evolves.

These varieties of conceptual models are sometimes distinguished in existing work, with different terminology. In 2013, Robinson used “conceptual model” to mean “a non-software specific description of the simulation model, … describing the objectives, inputs, outputs, content, assumptions, and simplifications of the [simulation] model” and “system description” to mean models derived from the “real world,” with two stages of computer-specific models derived from the system description (Robinson 2013). In a 2012 tutorial, Harrison and Waite use “conceptual model” to mean “an abstract and simplified representation of a referent (reality)” (Harrison and Waite 2012), instead of Robinson’s “system description.”

With this context, developing an engineering discipline of conceptual modeling will require much better understanding of:

1. how to make conceptual models explicit and unambiguous, for both the target system (or referent) and the target analysis,
2. the processes of conceptual modeling, including communication and decision-making involving multiple stakeholders,
3. architectures and services for building conceptual models.

Answering the first question (explicitness) requires considering alternative formalisms for expressing conceptual models, and the languages based on these formalisms, which are addressed in Sect. 3.1. The second question (process) is discussed in Sect. 3.2. The third question involves architectures for model engineering, as well as services provided to conceptual modelers, and is covered in Sect. 3.3.

E. Weisel
Old Dominion University, Norfolk, USA
e-mail: eweisel@odu.edu

L. Zhang
Beihang University, Beijing, China
e-mail: johnlin9999@163.com

3.2 Conceptual Modeling Language/Formalism

An articulated conceptual model, whether describing the system of interest (the *referent*, in Robinson's terminology) or an analysis model of the system of interest, is expressed using some language, which may be formal or informal, graphical, textual, mathematical, or logical. Today, the situation is that most often, conceptual models are expressed using some combination of sketches, flowcharts, data, and perhaps pseudo-code. Lack of general agreement on the implications of these techniques (i.e., ambiguity) limits the computational assistance that can be provided to engineers. Incorporating conceptual modeling into a modeling and simulation engineering discipline will require more explicit and formal conceptual modeling languages. However, conceptual modeling must be done in a manner accessible to domain engineers, who might not be trained in the necessary formalisms. This is addressed in the first subsection below. In addition, formal conceptual modeling applies as much to analysis as to the referent systems, raising questions about the variety of approaches to simulation, as covered in the second subsection. Formality in model integration is discussed in Sect. 3.3.

3.2.1 Domain-Specific Formalisms

In mathematical logic, formalism is the application of model and proof theory to languages, to increase confidence in inferring new statements from existing ones (Bock et al. 2006). In practice, however, most mathematicians are more informal in their definitions and proofs, with peer review confirming results, or not. We expect conceptual modeling formalisms to be rigorous approaches to studying referent and analysis models, at least in the sense of mathematical practice. Formal approaches have fewer, more abstract categories and terms than less formal ones, facilitating integration across engineering domains and construction of analysis tools. However, by using more abstract language, formal approaches are often too far from the common language of applications to be easily understood by domain experts and too cumbersome to use in engineering practice, e.g., in air traffic control, battlespace management, healthcare systems, and logistics. More specific formalisms would be useful not only to domain experts, for describing their systems, but also to technical or modeling experts who must translate the system description into analysis models and maintain them, and to other stakeholders who may need to participate in validation.

Logical modeling is a widely used approach to formalizing domain knowledge (often called *ontology*, more specifically description logics Baader et al. 2010). Ontologies can support acquisition of increasing levels of detail in model structure and also education and communication. For example, in modeling an ecosystem, one begins with words and phrases expressed in natural language, such as pond, organism, bio-matter, and insect. Some words will represent categories or classes,

while others represent instances falling into those categories. Also, words that connote action will reflect behaviors that are at the core of dynamic system specification. Words and phrases can be connected through relationships forming semantic networks and concept maps. Semantic networks (see, e.g., Reichgelt 1991) grew out of theories of cognition around associative memory (Quillian 1968), whereas concept maps (see Novak 1990) grew out of a theory of associative networks for the purpose of learning, both essential for capturing expert knowledge. Both are closely related to description logics (Sattler 2010; Eskridge and Hoffman 2012).

Developing explicit and formal conceptual models of the referent will require ontologies and a suitable knowledge representation. Contemporary modeling languages have been proposed and used for modeling software systems (UML OMG 2015b), for general systems modeling (OPM Reinhartz-Berger and Dori 2005; SysML OMG 2015a), and for modeling systems in the military domain (UPDM/UAF (OMG 2016), DoDAF (US Department of Defense 2016a), MoDAF (U.K. Ministry of Defense 2016)). Domain-specific modeling languages (DSMLs) also have been developed, for modeling business processes (OMG 2013), for modeling biological systems, SBML (Finney et al. 2006), SBGN (Le Novère et al. 2009), and others. Of course, general purpose languages can be specialized to a domain as well. In addition to using ontologies for domains, methods, and processes, DSMLs for representing knowledge along with induced constraints and interdependencies will also help to reduce uncertainty in the modeling process, e.g., to answer questions like what modeling approach, execution algorithm, or steady-state analyzer to use. Thus, ontologies and DSMLs for modeling and simulation methods are relevant definitely in the requirements stage, where it is decided which formalism to use and how to execute a model, and also for validating and verifying a large set of methods. Suitable ontologies, if in place, will help in identifying solutions.

While these developments are an important element of establishing an engineering discipline of modeling and simulation, they do not yet go far enough. Ontology is not sufficiently applied to formal and domain-specific modeling languages, leaving a major gap in linking formalisms to engineering domains. Many of the available models of formal and domain languages only categorize terminology rather than semantics of the terms, and consequently cannot utilize domain knowledge to increase the efficiency of formal computations or bring results from those computations back into the domains. For example, ontologies are available for Petri Nets (PN), a widely used simulation formalism, but these only formalize terminology for reliable interchange of PN models, rather than enabling a uniform execution of them across tools (Gaševića and Devedžić 2006). In addition, if adequate DSMLs do not already exist within a domain, each application modeler still must develop a problem-specific ontology and capture problem-specific knowledge in a DSML. Within a particular domain, e.g., logistics, the creation of a domain-specific ontology and modeling framework would support all modelers within that domain (Huang et al. 2008); (McGinnis and Ustun 2009); (Thiers and McGinnis 2011); (Batarseh and McGinnis 2012); (Sprock and McGinnis 2014).

Work on modeling formal and domain-specific languages, including semantics as well as terminology and how to integrate them for practical use, are in its early stages (Mannadiar and Vangheluwe 2010); (Bock and Odell 2011), but several results have emerged during the past decade. This is an important area for future research and development.

Language modeling (*metamodeling*) has become a widely used method for precisely defining the abstract syntax of DSMLs (the part of syntax that omits detailed visual aspects of a language). A metamodel is the model of a modeling language (Karsai et al. 2004), expressed by means of a metamodeling language (Flatscher 2002). There are several metamodeling languages in practical use today, ranging from informal, graphical languages, such as UML class diagrams and OCL used by the Object Management Group (OMG 2015b) (OMG 2014), the Eclipse Modeling Framework (Eclipse Foundation 2016a), or MetaGME (Emerson and Neema 2006). A formal metamodeling language based on algebraic datatypes and first order logic with fixpoint is FORMULA from Microsoft Research (Jackson and Sztipanovits 2009).

Metamodeling can be used to specify diagrammatic syntax of DSMLs, in conjunction with their abstract syntaxes above. For example, languages such as Eclipse's Graphical Modeling Project (Eclipse Foundation 2016b) and WebGME (Institute for Software Integrated Systems 2016) provide a graphical metamodeling environment, as well as auto-configuration into domain-specific modeling environments using the metamodels created there. Metamodeling of diagrammatic syntax also enables standardized interchange between tools and rendering of graphics (Bock and Elaasar 2016).

Metamodeling has a role in precisely defining the semantics of DSML's. For example, FORMULA's constraint logic programming capability is used for defining semantics of DSMLs via specifying model transformations to formal languages (Simko et al. 2013). The portion of UML's metamodel that overlaps description logic can be extended to specify patterns of using temporal relation models in UML, providing a basis to formalize the semantics of UML's behavioral syntaxes (Bock and Odell, 2011).

Other approaches to the development of DSMLs include developing a DSML for a subset of the Simulink language, by defining the operational semantics, rather than by creating a meta-model (Bouissou and Chapoutot 2012), or in a similar vein, developing a DSML for systems biology based on an abstract syntax and operational semantics (Warnke et al. 2015).

3.2.2 A Unified Theory for Simulation Formalisms

Conceptual modeling applies not only to the system of interest, but also to the analysis of that system. Our understanding of a system of interest evolves from our earliest concept of it as we gain deeper understanding through the development of system models. In the same way, our understanding of the analysis itself also may

evolve as we better understand the system of interest and begin to elaborate our analysis model. To support conceptual modeling of simulation analysis, it seems reasonable that we should first have the ontology, semantics, and syntax to formally define a simulation. Unlike the case of other analyses, such as optimization, this requirement has not yet been satisfied for simulation. Several structures have been studied as simulation formalisms; however, there is little consensus on the best approach. In the same way that various models of computation provide a basis for theory within computer science, considering various simulation formalisms will further the development of a robust theory of simulation.

Some formalisms are available for general discrete event simulation, some adopted industrially and others not. For example, the DEVS language (Zeigler et al. 2000) provides a mathematically precise definition of discrete event systems, and there also are a number of computational implementations, so it is unique in providing both a simulation programming language and an associated mathematical specification. It is not widely used industrially, however, in part, perhaps, because of the requirement to express all behavior using state machines. Popular discrete event simulation languages or environments, such as Arena (https://www.arenasimulation.com), FlexSim (https://www.flexsim.com), Simio (http://www.simio.com), and Tecnomatix Plant Simulation (https://goo.gl/XmQGgN), provide a programming language with semantics and syntax, but not a corresponding formal definition. In part, this is due to the intent of many commercial simulation languages to support simulation in a particular domain, such as Tecnomatix Plant Simulation, which is naturally reflected in the semantics of the languages.

Another line of research is to view simulations through the lens of dynamical systems and computational complexity theory. This is particularly suitable when studying complex socially coupled systems. Formal computational and mathematical theory based on network science and graphical dynamical systems has been studied in Mortveit and Reidys (2008), Barrett et al. (2004, 2006), Adiga et al. (2016), Rosenkrantz et al. (2015). The theoretical framework allows one to study formal questions related to simulations, including: (i) computational lower and upper bounds on computing phase space properties, (ii) design questions: how does one design simulations to achieve a certain property, (iii) inference questions: how does one understand the conditions that led to the observed behavior.

Achieving an engineering discipline for modeling and simulation will require a more complete set of formalisms spanning up from rigorous discrete event, continuous, and stochastic system specification to higher level, perhaps domain-specific, simulation languages. In some areas, those domain-specific modeling languages that combine a rigorous mathematical semantics with a convenient modeling tool are already in use, e.g., in the area of cell biology, or collective adaptive systems (often based on a continuous time Markov chain semantics). For example, some specialized simulation languages for biology are based on mathematical formalisms, such as ML-Rules (Helms et al. 2014), Kappa (Harvard Medical School 2016), or BioNetGen (BioNetGen 2016), among others. In general, however, this still represents a very significant challenge for the modeling and simulation community.

3.3 Conceptual Model Development Processes

Model development is a challenging and highly intricate process, with many questions needing to be answered, as discussed in this section. Currently, answering these questions in a systematic and informed manner is hampered by a lack of formalized knowledge in the modeling domains and in modeling and simulation in general. Providing these would constrain development decisions and the design of development processes themselves, reducing uncertainty in model life cycle engineering. The first subsection below gives background on model development processes and analyzes questions about them. The next two subsections (effectiveness and maturity) describe complementary approaches to reducing model defects introduced during the modeling process. These help avoid difficult and high-cost amendments of the model after it is finished. It is impossible to reduce model defects to zero during development, leading to the need for validation after the model is built, the results of which are also useful during model development, as addressed in the last subsection. Taken together, progress in these areas can significantly enhance the credibility of models by improving the quality of processes that produce them.

3.3.1 Motivation and Research Approach

The purpose of modeling and simulation is to improve our understanding of the behavior of systems: An executable model M of a system S together with an experiment E allows the experiment E to be applied to the model M to answer questions about S (Cellier 1991). Simulation is fundamentally an experiment on a model. A conceptual model C is the articulated description of S, upon which both M and E are developed. In science we seek to understand the behavior of natural systems; in engineering we seek to design systems that exhibit desired behavior. Because modeling and simulation facilities are themselves complex systems, it is seldom possible to go in one step from problem to solution. The processes involved in modeling and simulation require different degrees of human interaction, different computer resources, are based on heterogeneous, partly uncertain knowledge defined more or less formally, and involve different types of expertize and users. Data, knowledge, processes, and orchestration vary depending on the system to be modeled, the questions to be answered, and the users. In these processes different versions of models and artifacts are generated, that need to be put into relation to each other.

Model life cycle Engineering (MLE) captures the highly iterative process of developing, verifying, validating, applying, and maintaining a model. MLE is an area that requires significant study and exploration to meet society's needs and problems. How is MLE different than Engineering Design or Software Engineering life cycles? In some instances, it may be possible to build on these related

engineering fields in our attempt to forge MLE as a subdiscipline of Modeling and Simulation (M&S) . It is expected that MLE will contain phases for constructing models and simulations by beginning with requirements and then proceeding to other phases such as design, analysis, implementation, verification and validation (V&V), and maintenance.

MLE concepts and methods should not be limited to developing M and E; they also should be applied to the conceptual model C, describing S and used in developing both M and E. Clearly, this requires that C be expressed in a form that enables MLE concepts and methods to be applied.

The underlying principle for any type of life cycle engineering, however, is to ensure that unspent resources (e.g., money, time) are commensurate with work remaining. For complex systems with substantial de novo content, there is typically considerable uncertainty in both the work remaining and the rate of resource consumption. Resources are therefore held in reserve to protect against depletion due to undesired outcomes. Bearing these principles in mind, a life cycle approach for model/simulation development should include answering the following questions:

- Purpose and Scope Characterization: Who are the stakeholders of the model? What are their concerns? In particular, what are the specific aspects of system behavior we seek to understand through modeling? The answers form the context for the relevant conceptual models. Identification of stakeholders and concerns is a complex undertaking involving a broad spectrum of disciplines, including perhaps the political and behavioral sciences. For example, a macroeconomic simulation of energy production, distribution, and consumption would rightly recognize the public at large as a stakeholder, but it would be counterproductive to ask individual citizens simply to enumerate their concerns since ordinary citizens are not likely to understand the stake they have in atmospheric carbon dioxide or sulfur dioxide. Consequently, it may be necessary to develop methods that combine opinion research with education and outreach to designated proxies for the public interest.
- Phenomena Characterization: Is the referent a continuous system, discrete event system, or discrete stepwise system? Are stochastic or spatial aspects important? What are the elements of the system which contribute to the behavior of interest? What scientific disciplines address the behavior of interest? Answers to these kinds of questions will help to identify the content of the conceptual model and perhaps how it should be expressed. Having identified concerns, it is not necessarily simple to determine the scope of scientific phenomena to adequately address those concerns. For example, if stakeholders are concerned about the availability of drinking water, it may under some circumstances suffice to consider only hydrological phenomena. Under other circumstances it may be necessary to consider also social, economic, and political phenomena. Decisions will ultimately of course involve judgment, but research may elucidate principles and techniques that might prove useful for such analysis.

- Formalism Characterization: What formulations will be most appropriate to describe the relevant system elements and characterize the phenomena of interest in the form of input-output relations? The conceptual model must support these formulations. The choice of formalism will depend on the nature of the system being modeled, as determined by phenomena characteristics above. Once the nature of the system is identified, how is it best described, e.g., for a continuous system, are block diagrams most appropriate, or systems dynamics, or an object-oriented approach like Modelica (Modelica Association 2014b)? What mathematical formulations will be used to characterize the phenomena of interest in the form of input-output relations? Differential equations? Statistical models? Logical models? A given phenomenon may be mathematically characterized in different ways, depending upon, among other things, the nature of the concerns under consideration. If we are primarily concerned with long-term average behavior, we might choose a lumped-parameter description that assumes all short-term variation self-cancels over time. On the other hand, if we are concerned with infrequent extreme events, we will require a characterization that captures higher-order dynamics accurately. Research may help us better understand how to infer the possible mathematical formalism from a given referent model, but also how to develop requirements for the referent model from a useful mathematical formalism.
- Algorithm Characterization: What solution algorithms will be selected for computing the input-output relations? What verification test cases are appropriate? Since the conceptual model is a bridge from S to the computational model M, it may be important to understand and accommodate the specific target algorithmic implementation. For example, canonical linear least squares problems can be stated more compactly as so-called normal equations, but in practice, normal equations are more computationally complex to solve than the better-behaved and more efficient orthogonal triangularization. These are considerations of numerical analysis, a mature field with decades of sound theory and practical technology, but needing integration into M&S methodologies.
- Model Calibration: What data is available to calibrate and later validate the model, M? Is it necessary to calibrate a conceptual model, and if so, how is it done? How does one validate a conceptual model?
- Cross-validation: Do other models exist with which the new model can be cross-validated? If there are other existing conceptual models, how can they be compared to support cross-validation?

These questions need to be addressed during the requirements phase of the model engineering life cycle. However, answers are likely to be revised during the subsequent phases. From this point on, conventional software development life cycle considerations apply. In addition, special consideration needs to be given to validation and verification of model variants and their interdependencies. Research is needed to understand how to help answer the above questions: how to manage the evolution process of a model and the data, knowledge, activities, processes and organizations/people involved in the full life cycle of a model?

Managing the life cycle process of a model is one of the most important tasks of model engineering. Some research topics should be attacked, for example, how to structurally describe the modeling process, and how to identify the characteristics of activities involved in model construction and management to ensure improvement of model quality and development efficiency and reduction of full model life cycle cost.

Some decisions, e.g., which execution algorithm to select, might even be supported automatically by exploiting machine learning methods (Helms et al. 2015). However, automatic solutions to these decisions require metrics to clearly distinguish the good choices from the less suitable ones. For some decisions, e.g., selecting the modeling approach, providing suitable metrics is still an open challenge. Knowledge about constraints on applying one method or the other, and interdependencies and implications of using one or the other method on future activities will reduce uncertainties in the overall process.

Within the engineering of models, well-founded answers to the questions of which step to do next and which method to use largely determine the efficiency and effectiveness of the model engineering process. Referring to the first question, and for orchestrating the diverse processes that are involved in modeling engineering, workflow-based approaches might be exploited to make these processes explicit and traceable. These approaches facilitate evaluation of different phases of model life cycle engineering, including validation and verification of models, and, thus, add to the credibility of M&S. However, this requires a high degree of standardization of these processes. This might be achievable for specific subprocesses of validation or verification, e.g., how to execute and analyze a parameter scan given a specific model. However, the overall process of a simulation study is highly interactive and thus one might only be able to define general constraints on the engineered artifacts, e.g., if the conceptual model (if we interpret conceptual model as a representation of requirements or invariants that refer to the simulation model) changes, so does the stage of the process model, requiring a new validation phase.

3.3.2 Effectiveness Measures

In a model-based engineering (MBE) approach, the development team evolves a set of models throughout the system life cycle to support design, analysis, and verification of the system under development. These models are intended to express aggregate knowledge about the system to enable communications and shared understanding among the development team and other program stakeholders. Program leadership must continue to determine what knowledge must be acquired at any given point in the life cycle to maximize the likelihood of program success. The type of knowledge to be acquired can help identify the kind of design and analysis models that should be further developed and updated.

This knowledge can be acquired by performing engineering tasks that involve different kinds of models, such as performing a trade study to select among

alternative system architectures, performing an analysis to determine a system error budget, updating electrical, mechanical, or software designs, or analyzing a particular design for reliability, safety, manufacturability, or maintainability. Determining what knowledge is needed becomes more challenging as the complexity of the system increases, and as the complexity of the organization that develops the system increases (e.g., large geographically distributed teams).

The research challenge is to define one or more effectiveness measures that can guide the knowledge acquisition process and associated model development and evolution throughout the system life cycle. In other words, how do you determine the additional knowledge at each point in time that provides best value to the program stakeholders? The research can benefit from data that has been collected over many years to find a solution. For example, the following figures are typical examples of trends that indicate the impact of collecting certain kinds of knowledge on the overall cost of system development.

In Fig. 3.1 the lower curves reflect the percentage of the total life cycle cost that is expended as a function of the phase of the program life cycle. As indicated, much of the cost is expended in the later life cycle phases. However, as shown in the upper curve, the percentage of the life cycle cost that is committed occurs much earlier in the life cycle. This finding shows the importance of early design decisions based on the available knowledge. The cost to fix a defect increases exponentially as a function of the phase in the product life cycle where the defect is detected (Boehm 1981; McGraw 2006). Acquiring the knowledge to surface defects early can substantially reduce the total system life cycle cost.

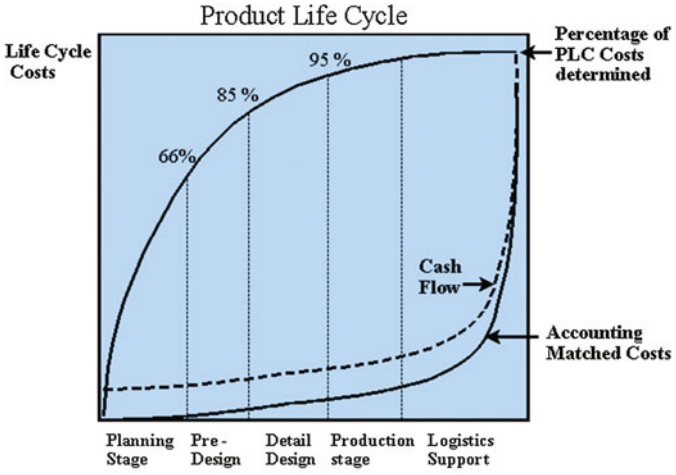

Fig. 3.1 Committed and actual lifecycle costs (Berliner and Brimson 1988) citing (Blanchard 1978)

The following are some suggested factors to be considered for research:

- Aggregate knowledge goals at particular milestones in a system development life cycle.
- Knowledge elements that contribute to the aggregate knowledge.
- Knowledge elements associated with different aspects of the system of interest and its environment.
- A value function associated with acquiring knowledge elements at each point in time, and its impact on the probability of program success.
- Cost to acquire the knowledge elements at a given point in the life cycle.
- Cost associated with acquiring incorrect knowledge at a given point in the life cycle.
- Relationship between the effectiveness measure (value vs. cost) and more traditional risk measures.

The acquisition of knowledge across a life cycle can be thought of as a trajectory whose aim is to maximize program success. The value function of acquired knowledge is dependent on both the knowledge elements and the sequence in which these elements are acquired, since there are dependencies among the knowledge elements. For example, during the concept phase of a vehicle's development, it is often important to acquire knowledge about vehicle sizing and system level functionality to meet mission performance requirements, but it may not be important to acquire knowledge about the detailed software design algorithms.

3.3.3 Maturity Models

The Capability Maturity Model (CMM) for software development has played a key role to guarantee the success of software projects (Paulk et al. 1993). CMM and CMM Integration (CMMI) originated in software engineering, but have been applied to many other areas over the years (CMMI 2016a). However, in M&S, there is no such standardized and systematic assessment methodology developed for M&S processes. Some related research and development results can be used as references to establish the maturity model of M&S:

- Software life cycle models describe core processes for software development. Following proven processes for model development begins with an understanding and execution of the core activities in an organization's chosen development path. Software life cycle models are an example of core proven processes for development. Whether the life cycle model chosen is the classic waterfall model or more modern iterative versions, all have aspects of requirements development, design, implementation, integration, test, etc., utilized in a way that best fits the size of the organization, the size of the project, or the constraints of the customer and developer.

- Software CMMI or CMMI for Development shows the success of maturity for general software development. CMMI was originally developed at Carnegie Mellon University and the federally funded Software Engineering Institute (SEI). The CMMI Institute reports that thousands of CMMI appraisals are completed every year in dozens of countries (CMMI 2016b). CMMI enables organizations to be viewed and certified as being mature and capable of carrying out intended activities to a certain level or degree of expertise, which lends that degree of credibility to the components developed by those activities. CMMI assigns *capability* levels to process improvement achievement in individual process areas. Therefore, a certain part of an organization may be identified or certified at a level 3 out of 4 for Configuration Management, but capability level 2 out of 4 for Maintenance. CMMI assigns *maturity* levels to process improvement achievement in multiple process areas or a set of process areas and applies to the scope of the organization that was evaluated/certified such as a department, a division, or the entire company—Level 5 being the highest achievable level of maturity.
- The Federation Development and Execution Process (FEDEP) describes core processes for simulation development. FEDEP was initially released in 1996 as the first common process for the development of simulations and was specifically for guidance in creating High Level Architecture (HLA) federations (IEEE Standards Association 2010). These common methodologies and procedures consisted of six steps: (1) Define Objectives, (2) Develop Conceptual Model, , (3) Design Federation, (4) Develop Federation, (5) Integrate and Test, and (6) Execute and Prepare Results. These six steps included specific work products that were inputs and outputs to each step. These steps and the FEDEP process paralleled the software development process and could serve as the initial draft of the core processes for model and simulation development that would be the basis for an examination of an organization's ability to robustly, reliably, and repeatedly develop credible models and simulations, i.e., identify the capability and maturity of organizations or portions of organizations.

System of Systems describes corollaries that may exist within Systems Engineering (Zeigler and Sarjoughian 2013).

- DoDAF describes the notion of multiple views of simulations (US Department of Defense 2016b).

Taking CMM/CMMI as a basis, a capability maturity model for modeling and simulation process (MS-CMMI) could be established by:

- Finding the differences and similarities between the processes of modeling/simulation and software development by analyzing characteristics of modeling process and simulation of complex systems, then define indicators and metrics for M&S processes.

- Setting up a MS-CMMI evaluation system (evaluation methods, standards, tools, organizations, etc.) to assess the structured level of capabilities of model developers or model users (use the model to do simulation).

Achieving these goals requires research in:

- Quantitative analysis of the complexity and uncertainties in modeling processes.
- Optimization of modeling processes.
- Risk analysis and control of modeling processes.
- Quantitative measurement of model life cycle quality and cost.
- Notional mappings with CMMI, etc.
- Identification and description of processes and work products necessary at differing levels of the Modeling and Simulation Maturity Model, when and why they are needed and who performs them.

3.3.4 Validation

As simulation models become more complex, validation of conceptual models and understanding their role in the broader process of validation will continue to be important research areas. Of course, understanding validation of conceptual models is dependent on a precise definition of the terms "conceptual model" and "validation." This section argues that a better consensus is needed on the first term, while a careful review of the validation literature will reveal the same for the second.

This is particularly apparent across M&S communities of practice. For example, the training and engineering communities intersect the broader M&S community, but M&S stakeholders in those communities draw heavily from skill sets based in different scientific disciples and different perspectives of the role of modeling and simulation. The M&S community's challenge is to address universally applicable concepts, like conceptual models in validation, from a holistic perspective with theory that is satisfying to all the stakeholders and technology that is germane to a broad set of problems (in the case of the stated example, simulation theory and technology that is useful for both the social scientist and the engineer).

Consider a few simple questions. What does it mean to validate a conceptual model? How does a conceptual model that is suitable for a specific use inform the development of other simulation process artifacts? How do the various stakeholders in the simulation activity use the conceptual model, valid or otherwise? Some researchers will see these as easily answered in their particular domains, but will find their conclusions quite different between domains. So, for the discussion in this section we consider terminology in the broadest context possible.

Consider modeling paradigms as equivalence classes on the set of conceptual models. Each paradigm has defining characteristics in terms of conceptual modeling language or formalism. These characteristics define every instance of a conceptual model as belonging to one class or another, or perhaps none. For well-developed

theory, further properties and theorems will follow to enable reasoning on all of the elements of each class in general without resorting to building, coding, and executing every instance to understand its properties. As we develop more rigorous and explicit conceptual models that bridge between the referent system and the computer simulation, methods for validation will become even more critical.

Simulation frameworks that include a category for conceptual models permit the side-by-side comparison to and facilitate discussion of related artifacts. For example, Balci and Ormsby (2007), Petty (2009), and Sargent (2013) provide frameworks that include conceptual models in this context. Advances in conceptual modeling will drive the need for new frameworks to explain the properties of conceptual models, and the relations between them, the referent system, and the computer simulation.

Some researchers would consider that the conceptual model is an appropriate artifact to analyze for suitability for use. Although recent work in validation theory is looking hard at the implication of risk in the decision to use particular kinds of simulation, and propagation of error in simulation, more basic research is needed to develop a robust model-based decision theory. Accuracy is well understood, particularly in the context of physics-based models, but its use in simulation is not well defined. When deciding on the kind of simulation to inform a particular decision, acceptability criteria are often subjective and little theory exists to objectify the decision analysis. A well-developed model-based decision theory will recast validation in the language of decision theory, defining use in a rigorous way, clearly differentiating objective from subjective elements of the use decision, and providing a defensible basis for using models and simulations to inform decision-making (Weisel 2012).

The logical next step for advances in theory involving validation of conceptual models is to incorporate these advances in simulation development environments. As model life cycle engineering develops, tools for validation of conceptual models are needed to keep pace. As conceptual modeling languages and formalisms become useful additions to simulation development environments, tools using well-defined conceptual models within the broader process of validation will improve the quality and defensibility of the simulation end product.

It is well understood that validation is best considered early in the development process (see the previous two subsections)—there should be no difference when considering conceptual models in the mix. As development environments would benefit from rigorous application of conceptual models in the development process, so too would the consideration of validation from the earliest life cycle stages. New technologies and tools are needed to incorporate validation of conceptual models throughout the simulation life cycle.

3.4 Conceptual Model Architecture and Services

Many modeling paradigms exist for most kinds of domain problems, applied to knowledge from many engineering disciplines. Understanding complex systems requires integrating these into a common composable reasoning scheme (NATO Research and Technology Organization 2014). The software and the system engineering communities have overcome similar challenges using architecture frameworks (e.g., OMG's Unified Architecture Framework OMG 2016), but modeling and simulation does not have a similarly mature integration framework. The first subsection below concerns architectures for conceptual modeling, while the second outlines infrastructure services needed to support those architectures.

3.4.1 Model Architecture

At the foundation of a modeling architecture should be a fundamental theory of models, to enable reusability, composability, and extensibility. What theory of models could support the implementation of a model architecture? An epistemic study of existing modeling and integration paradigms is necessary to develop a theory of models. This should include a taxonomy of modeling paradigms, semantics, syntaxes, and their decomposition into primitives that operate under common rules across paradigms, to integrate them as required by complex systems.

Model architecture is needed to unify different classes of models developed using different paradigms. An architecture is the glue specifying interfaces, rules of operation, and properties common across modeling paradigms, enabling models to be interconnected at multiple levels of conceptual abstraction. What is meaningful to connect? What is not? An architecture goes far beyond conventional model transformations and gateways, though these are also essential to comprehension of multiparadigm modeling processes. An architecture is about persistent coexistence and coevolution in multiple domains at multiple levels of abstraction. How can a model architecture framework connect models that operate according to different sets of laws? For example, critical infrastructure protection requires connecting country, power grid, internet, economy, command, and control, etc. Combat vehicle survivability requires connecting humans, materials, optics, electromagnetics, acoustics, cyber, etc. What mechanisms are required to efficiently interact between different sets of laws (e.g., layered architecture)? What level of detail is required to observe emerging behaviors between different sets of laws when integrated? How should a model architecture be implemented, in which format, using which tools? As a model architecture matures, successful design patterns should emerge for the most common reusable interconnections between disciplines. What are these design patterns in each community of interest?

Model architecture sets the rules to meaningfully interconnect models from different domains. Generalizing and publishing rules for widespread modeling

paradigms would allow composing and reusing models that comply with the architecture and complex system simulations will become achievable. As an example of interconnected models across domains, start with a Computer-Aided Design (CAD) model representing a physical 3D object in terms of nodes and facets. In the CAD paradigm, objects can be merged to interconnect. A related Finite Element Model (FEM) represents continuous differential equations for physical laws between boundary layers. It can be used to compute the fluid dynamics during combustion. FEM models can interconnect at the level of physical laws to compute the temperature distribution from the combustion products distribution for instance. They also interconnect with a CAD model at the mesh level. A computer graphics model enables display of objects as seen from particular viewpoints. It interconnects with CAD and FEM models to map materials and temperature to facets for the purpose of generating an infrared scene image in the field of view of a sensor. A functional model of a surveillance system can represent discrete events involved in changing a sensor mode as a function of the mission. The functional model interconnects with the computer graphics model at the sensor parameter level. Finally, a business process model can represent a commander's mission planning. It can interconnect with a functional model by changing the mission.

Figures of merit must be developed to demonstrate how well a model architecture facilitates composition of multiparadigm, multiphysics, multiresolution models. The performance of a model architecture must be checked against interdisciplinary requirements using metrics for meaningfulness and consistency. How can we test a particular integration for validity? How can it be done efficiently over large-scale complex simulations? How can it be done by a non-expert? What mechanisms should a model architecture framework include to support checking for conceptual consistency?

Integration complexity and coupling between the degrees of freedom of individual components and the degrees of freedom of the integration are yet to be understood. When integrating a model in a complex simulation, what details can be ignored and still ensure a valid use of that model? What details cannot be ignored?

Reliable model integration depends on sufficient formality in the languages used, as described in Sect. 3.1. In particular, formal conceptual models of both the system of interest (referent) and analysis provide a basis for automating much of analysis model creation through model-to-model transformation. As an example, consider the design of a mechanical part or an integrated circuit. The CAD tools for specifying these referents use a standard representation, with a formal semantics and syntax. For particular kinds of analyses—such as response in an integrated circuit—simulations are essentially available at the push of a button. Formalism in the specification of the referent enables automation of certain analyses. This pattern is well demonstrated, e.g., in the use of Business Process Modeling Notation (BPMN) to define a business process, and then automating the translation of this model into a hardware/software implementation specification. The Object Management Group has developed standard languages for model-to-model transformations. At present, there are only limited demonstrations of applying this approach to systems

modeling. Automating this kind of model-to-model transformation captures knowledge about how to create analysis models from referent models, so perhaps the most fundamental question is: where should this knowledge reside—should it be captured in the referent modeling language, in the analysis modeling language, in the transformation, or perhaps spread throughout? Formalization of mappings between conceptual models of a referent and its analysis models is critical to building reliable bridges between descriptions of the referent and specifications of a simulation model and its computational implementation.

3.4.2 Services

The success of large-scale integration of knowledge required by complex systems fundamentally depends on modeling and simulation infrastructure services aggregated into platforms. These enable affordable solutions based on reusing domain-specific models and simulators, as well as integrating them into a multi-model cosimulation. For example, understanding vulnerabilities and resilience of complex engineered systems such as vehicles, manufacturing plants, or electric distribution networks requires the modeling and simulation-based analysis of not only the abstracted dynamics, but also some of the implementation details of networked embedded control systems. Systems of such complexity are too expensive to model and analyze without reuse and synergies between projects.

Services need to enable open model architecture development and sharing of model elements at all levels. How can a common conceptual modeling enterprise be launched involving many stakeholders? How can a conceptual model be augmented with knowledge from different contributors (e.g., wiki)? How does it need to be managed? What structure should the conceptual model have? What base ontologies are required (e.g., ontology of physics)? How can conceptual model components be implemented in executable model repositories and how can components plug and play into simulation architectures? Guiding principles must also be defined and advertised. What guidance should modelers follow to be ready for a collaborative conceptual modeling enterprise in the future? Standard theory of models, architecture, design patterns, consistency tests, modeling processes, and tools will arise naturally as the modeling science matures.

Services can be aggregated into three horizontal integration platforms:

- In *Model Integration Platforms*, the key challenge is to understand and model interactions among a wide range of heterogeneous domain models in a semantically sound manner. One of the major challenges is semantic heterogeneity of the constituent systems and the specification of integration models. Model integration languages have become an important tool for integrating complex, multimodeling design automation and simulation environments. The key idea is to derive opportunistically an integration language that captures only

the cross-domain interactions among (possibly highly complex) domain models (Cheng et al. 2015).

- *Simulation Integration Platforms* for cosimulation have several well-established architectures. The High Level Architecture (HLA) (IEEE Standards Association 2010) is a standardized architecture for distributed computer simulation systems. The Functional Mockup Interface (Modelica Association 2014a) for cosimulation is a relatively new standard targeting the integration of different simulators. In spite of the maturity and acceptance of these standards, there are many open research issues related to scaling, composition, large range of required time resolution, hardware-in-the-loop simulators and increasing automation in simulation integration.
- *Execution Integration Platforms* for distributed cosimulations are shifting toward cloud-based deployment, developing simulation as a service using models via web interfaces and increasing automation in dynamic provisioning of resources as required. More will be said about this in the next chapter.

References

Adiga, A., C. Kuhlman, M. Marathe, S. Ravi, D. Rosenkrantz, and R. Stearns. 2016. Inferring local transition functions of discrete dynamical systems from observations of system behavior. *Theoretical Computer Science*.

Baader, F., D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider (eds.). 2010. The description logic handbook: Theory, implementation and applications, 2nd ed.

Balci, O. and W. F. Ormsby. 2007. Conceptual modelling for designing large-scale simulations. *Journal of Simulation* 1: 175–186.

Barrett, C., S. Eubank, V. Kumar, and M. Marathe. 2004. Understanding large scale social and infrastructure networks: a simulation based approach. SIAM news in Math Awareness Month on The Mathematics of Networks.

Barrett, C., S. Eubank, and M. Marathe. 2006. Modeling and simulation of large biological, information and socio-technical systems: an interaction based approach. *Interactive computation*, 353–394. Berlin Heidelberg: Springer.

Batarseh, O. and L. McGinnis. 2012. System modeling in SysML and system analysis in Arena. In *Proceedings of the 2012 Winter Simulation Conference*.

Berliner, C. and J. Brimson, (eds.). 1988. Cost management for today's advanced manufacturing: the CAM-I conceptual design. Harvard Business School Press.

BioNetGen. 2016. "BioNetWiki." Accessed 29 Aug 2016. http://bionetgen.org.

Blanchard, B. 1978. *Design and manage to life cycle cost*. Dilithium Press.

Bock, C., M. Gruninger, D. Libes, J. Lubell, and E. Subrahmanian. 2006. *Evaluating reasoning systems*. Report: U.S National Institute of Standards and Technology Interagency. 7310.

Bock, C., and J. Odell. 2011. Ontological behavior modeling. *Journal of Object Technology* 10 (3): 1–36.

Bock, C., M. Elaasar. 2016. Reusing metamodels and notation with Diagram Definition. *Journal of Software and Systems Modeling*.

Boehm, B. 1981. *Software engineering economics*. Prentice-Hall.

Bouissou, O. and A. Chapoutot. 2012. An operational semantics for Simulink's simulation engine. Languages, compilers, tools and theory for embedded systems. In *Proceedings of the 13th ACM SIGPLAN/SIGBED International Conference, (LCTES'12)*. 129–138.

Cellier, F. 1991. *Continuous systems modelling*. Springer.
Cheng, B., T. Degueule, C. Atkinson, S. Clarke, U. Frank, P. Mosterman, and J. Sztipanovits. 2015. Motivating use cases for the globalization of CPS. In *Globalizing Domain-Specific Languages*, LNCS, vol. 9400, 21–43. Springer.
CMMI Institute. 2016a. "CMMI Models." Accessed 29 Aug 2016. http://cmmiinstitute.com/cmmi-models.
CMMI Institute. 2016b. 2015 Annual Report to Partners, Accessed 6 Sept 2016. http://partners.cmmiinstitute.com/wp-content/uploads/2016/02/Annual-Report-to-Partners-2015.pdf.
Eclipse Foundation. 2016a. Eclipse modeling framework. Accessed 29 Aug 2016. https://eclipse.org/modeling/emf.
Eclipse Foundation. 2016b. Graphical modeling project. Accessed 29 Aug 2016. http://www.eclipse.org/modeling/gmp.
Emerson, M., S. Neema, and J. Sztipanovits. 2006. *Metamodeling languages and metaprogrammable tools*. Handbook of Real-Time and Embedded Systems: CRC Press.
Eskridge, T., and R. Hoffman. 2012. Ontology creation as a sensemaking activity. *IEEE Intelligent Systems* 27 (5): 58–65.
Finney, A., M. Hucka, B. Bornstein, S. Keating, B. Shapiro, J. Matthews, B. Kovitz, M. Schilstra, A. Funahashi, J. Doyle, and H. Kitano. 2006. Software infrastructure for effective communication and reuse of computational models. In *System Modeling in Cell Biology: From Concepts to Nuts and Bolts*. Massachsetts Institute of Technology Press.
Flatscher, R. 2002. Metamodeling in EIA/CDIF—meta-metamodel and metamodels. *ACM Transactions on Modeling and Computer Simulation.* 12 (4): 322–342.
Gaševića, D., and V. Devedžić. 2006. Petri net ontology. *Knowledge-Based Systems* 19 (4): 220–234.
Harrison, N., W.F. Waite. 2012. Simulation conceptual modeling tutorial. In *Summer Computer Simulation Conference*, July 2012.
Harvard Medical School. 2016. KaSim: A rule-based language for modeling protein interaction networks. Accessed 29 Aug 2016. http://dev.executableknowledge.org.
Helms, T., C. Maus, F. Haack, and A. M. Uhrmacher. 2014. Multi-level modeling and simulation of cell biological systems with ML-rules: a tutorial. In *Proceedings of the Winter Simulation Conference*, 177–191.
Helms, T., R. Ewald, S. Rybacki, A. Uhrmacher. 2015. Automatic runtime adaptation for component-based simulation algorithms. *ACM Transactions on Modeling and Computer Simulation*, 26 (1).
Huang, E., Ky Sang Kwon, and L. F. McGinnis. 2008. Toward on-demand wafer fab simulation using formal structure and behavior models. In *Proceedings of the 2008 Winter Simulation Conference*.
IEEE Standards Association. 2010. 1516–2010-IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA). http://standards.ieee.org/findstds/standard/1516-2010.html.
Institute for Software Integrated Systems. 2016. "WebGME." Accessed 29 Aug 2016. https://webgme.org.
Jackson, E., and J. Sztipanovits. 2009. Formalizing the structural semantics of domain-specific modeling languages. *Journal of Software and Systems Modeling* 8: 451–478.
Karsai, G., M. Maroti, A. Lédeczi, J. Gray, and J. Sztipanovits. 2004. Composition and cloning in modeling and meta-modeling. *IEEE Transactions on Control System Technology.* 12 (2): 263–278.
Le Novère, N., M. Hucka, H. Mi, S. Moodie, F. Schreiber, A. Sorokin, E. Demir, K. Wegner. M. Aladjem, S. Wimalaratne, F. Bergman, R. Gauges, P.Ghazal, H. Kawaji, L. Li, Y. Matsuoka, A. Villéger, S. Boyd, L. Calzone, M. Courtot, U. Dogrusoz, T. Freeman, A. Funahashi, S. Ghosh, A. Jouraku, A, S. Kim, F. Kolpakov, A. Luna, S. Sahle, E. Schmidt, S. Watterson, S., G. Wu, I. Goryanin, D. Kell, C. Sander, H. Sauro, J. Snoep, K. Kohn, and H. Kitano. 2009. The systems biology graphical notation. *Natural Biotechnolology*. 27 (8): 735–741.
Mannadiar, R. and H. Vangheluwe. 2010. Domain-specific engineering of domain-specific languages. In *Proceedings of the 10th Workshop on Domain-Specific Modeling*.

McGinnis, L. and V. Ustun. 2009. A simple example of SysML driven simulation. In *Proceedings of the 2009 Winter Simulation Conference*.

McGraw, G. 2006. *Software Security: Building Security In*. Addison-Wesley.

Modelica Association. 2014a. Functional markup interface. https://www.fmi-standard.org/downloads#version2.

Modelica Association. 2014b. Modelica® -a unified object-oriented language for systems modeling, language specification, version 3.3, revision 1. Accessed 6 Sept 2016. https://www.modelica.org/ documents/ModelicaSpec33Revision1.pdf.

Mortveit, H., and C. Reidys. 2008. *An Introduction to Sequential Dynamical Systems*. Springer.

NATO Research and Technology Organisation. 2012. Conceptual Modeling (CM) for Military Modeling and Simulation (M&S). Technical Report TR-MSG-058, https://www.sto.nato.int/publications/STO%20Technical%20Reports/RTO-TR-MSG-058/$$TR-MSG-058-ALL.pdf, July 2012.

Novak, J. 1990. Concept maps and Vee diagrams: two metacognitive tools for science and mathematics education. *Instructional Science* 19: 29–52.

Object Management Group. 2013. Business process model and notation. http://www.omg.org/spec/BPMN.

Object Management Group. 2014. Object constraint language. http://www.omg.org/spec/OCL.

Object Management Group. 2015a. Systems modeling language. http://www.omg.org/spec/SysML.

Object Management Group. 2015b. Unified modeling language. http://www.omg.org/spec/UML.

Object Management Group. 2013/2016. Unified architecture framework. http://www.omg.org/spec/UPDM, http://www.omg.org/spec/UAF.

Paulk, M., W. Curtis, M. Chrissis, and C. Weber. 1993. Capability maturity model, version 1.1. Technical Report Carnegie Mellon University Software Engineering Institute. CMU/SEI-93-TR-024 ESC-TR-93–177, February 1993.

Petty, M. D. 2009. Verification and validation. In *Principles of Modeling and Simulation: A Multidisciplinary Approach*. John Wiley & Sons, 121–149.

Quillian, M. 1968. Semantic memories. In *Semantic Information Processing*. Massachusetts Institute of Technology Press, 216–270.

Reichgelt, H. 1991. *Knowledge Representation: An AI Perspective*. Ablex Publishing Corporation.

Reinhartz-Berger, I., and D. Dori. 2005. OPM vs. UML–experimenting with comprehension and construction of web application models. *Empirical Software Engineering*. 10 (1): 57–80.

Robinson, S. 2013. Conceptual modeling for simulation. In *Proceedings of the 2013 Winter Simulation Conference*.

Rosenkrantz, D., M. Marathe, H. Hunt III, S. Ravi, and R. E. Stearns. 2015. Analysis problems for graphical dynamical systems: a unified approach through graph predicates. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems (AAMAS'15). International Foundation for Autonomous Agents and Multiagent Systems*, Richland, SC, 1501–1509.

Sargent, R.G. 2013. Verification and validation of simulation models. *Journal of Simulation* 7: 12–24.

Sattler, U., D. Calvanese, and R. Molitor. 2010. Relationships with other formalisms, 149–193 (Baader et al. 2010).

Simko, G., D. Lindecker, T. Levendovszky, S. Neema, and J. Sztipanovits. 2013. Specification of cyber-physical components with formal semantics–integration and composition. In *Model-Driven Engineering Languages and Systems*. Springer Berlin Heidelberg, 471–487.

Sprock, T. and L. F. McGinnis. 2014. Simulation model generation of Discrete Event Logistics Systems (DELS) using software patterns. In *Proceedings of the 2014 Winter Simulation Conference*.

Thiers, G. and L. McGinnis. 2011. Logistics systems modeling and simulation. In *Proceedings of the 2011 Winter Simulation Conference*.

U.K. Ministry of Defense. 2016. MOD architecture framework. Accessed 29 Aug 2016. https://www.gov.uk/guidance/mod-architecture-framework.

U.S. Department of Defense. 2016a. The DoDAF architecture framework version 2.02. Accessed 29 Aug 2016. http://dodcio.defense.gov/Library/DoD-Architecture-Framework.

U.S. Department of Defense. 2016b. DoDAF viewpoints and models. Accessed 29 Aug 2016. http://dodcio.defense.gov/Library/DoD-Architecture-Framework/dodaf20_viewpoints.

Warnke, T., T. Helms, and A. M. Uhrmacher. 2015. Syntax and semantics of a multi-level modeling language. In *Proceedings of the 3rd ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, 133–144.

Weisel, E. W. 2012. A decision-theoretic approach to defining use for computer simulation. In *Proceedings of the 2012 Autumn Simulation Multi-Conference*.

Zeigler, B., H. Praehofer, and T. Kim. 2000. *Theory of Modeling and Simulation*, 2nd ed. Academic Press.

Zeigler, B., and S. Sarjoughian. 2013. *Guide to Modeling and Simulation of Systems of Systems*. London: Springer.